# DMN Tutorial

## DMN Online Simulator

Learning by Doing: You can use our free online simulator to execute DMN decision tables that you create.

DMN Online Simulator (https://camunda.com/dmn/simulator)

### Why should I care about DMN?

Decision Model and Notation (DMN) is an industry standard for modeling and executing decisions that are determined by business rules.

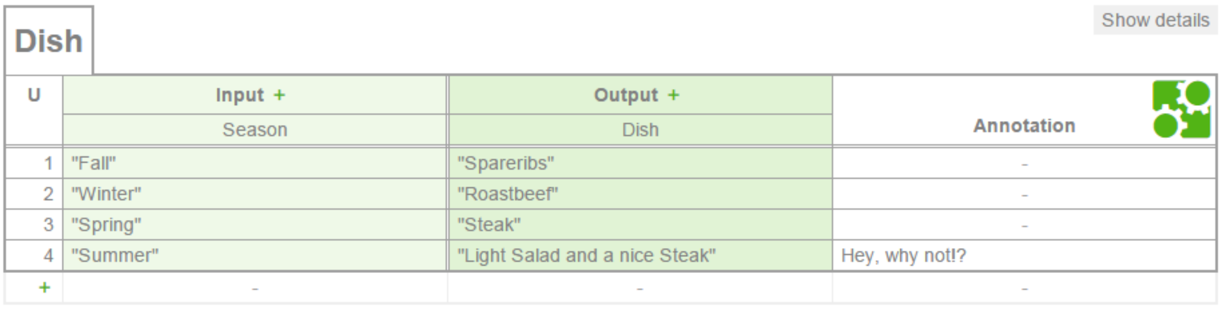
DMN has been published in 2015 and is currently seeing a very quick adoption. These are the reasons:

**Standard**
DMN is not owned by a certain enterprise but by an institution (OMG (http://www.omg.org)), which is already established through other world-wide standards, e.g., BPMN and UML. The DMN standard is supported by several software products; you are less dependent on any particular vendor's products.
**Direct Execution**
In DMN, decisions can be modeled and executed using the same language. Business analysts can model the rules that lead to a decision in easy to read tables, and those tables can be executed directly by a decision engine (like Camunda). This minimizes the risk of misunderstandings between business analysts and developers, and it even allows rapid changes in production.
**Experience**
DMN as a standard is young, but it was developed by people having decades of experience with business rule management. Even though, the standard does not dictate any special implementation patterns, allowing for more modern and lightweight implementations than traditional business rule engines.

This tutorial provides a quick introduction into DMN, as it is defined in version 1.1.

### A simple decision table

We should begin our DMN tutorial with a rather simple decision table:


Show details

Dish

| U | Input + | Output + | Annotation |
|---|---|---|---|
| | Season | Dish | |
| 1 | "Fall" | "Spareribs" | - |
| 2 | "Winter" | "Roastbeef" | - |
| 3 | "Spring" | "Steak" | - |
| 4 | "Summer" | "Light Salad and a nice Steak" | Hey, why not!? |
| + | - | - | - |

Let's assume we have invited some guests for dinner. The question is, which dish we should prepare. In this example, we follow a very simple decision logic: Depending on the current season, we decide on the dish. If it's Fall, we will go for spareribs, in Winter for roastbeef and so on.

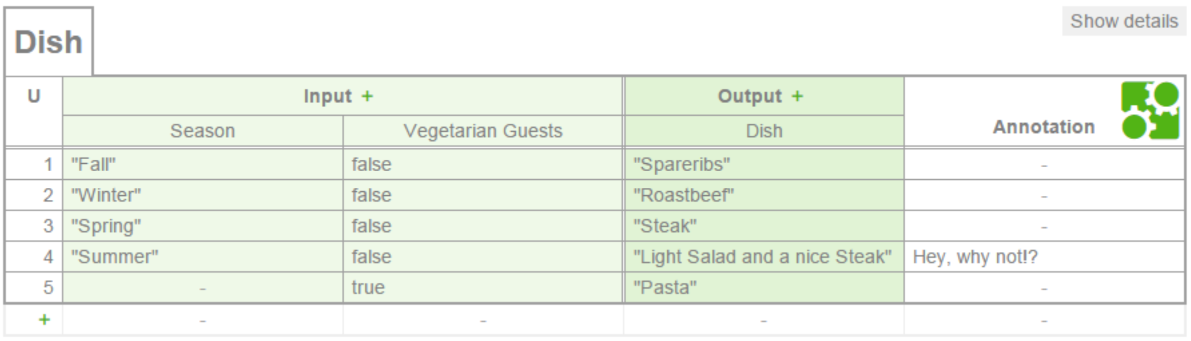
Let's look at the elements in this example:

- In the upper left corner we find the **name** of this decision table: "Dish"
- Below that is an "U", which stands for **unique** and is the defined **hit policy** for this decision table. It means that, when a decision has to be made, only one of the rows below can be true. In this case, the current season can only be fall, winter, spring or summer. We cannot have two seasons at the same time, even if the summer is bloody cold this year.
- The columns in light green refer to possible **input** data. In this example, there is only one input column, because we are only interested in the current season. The cell with the text "Season" defines this. In DMN, this is the label for an **input expression**. The expression itself is hidden in this example for the sake of simplicity, but will be revealed later in this tutorial. The cells below (called **input entries**) refer to the possible conditions regarding the input. Those conditons are in quotation marks (like "Summer"), which is because we are technically comparing String values.
- For each possible input entry (i.e. the name of the current season), we define the according **output entry** in the cell next to it. That's how we express that based on the season, we desire a certain dish. Again, we have to use quotation marks (like "Steak") because technically we are assigning String values.
- The definition, that according to an input entry that is true (or a combination of true input entries), a specific output entry should apply, is a **rule**. Each rule is defined in a table row below the table header, and has a number, which you can find in the cells to the left.
- Last but not least, you can **annotate** your rules in the column on the right. Those annotations are only there for the sake of explanation, and will be ignored by a decision engine.

Simple enough, isn't it? Of course there is more to DMN, but the basic principles are indeed very straight forward.

## Combining Conditions

In many cases a rule will not only consist of one condition, but a combination of conditions. We can express that by adding input columns to the decision table:


Show details

Dish

| U | Input + | | Output + | Annotation |
|---|---|---|---|---|
| | Season | Vegetarian Guests | Dish | |
| 1 | "Fall" | false | "Spareribs" | - |
| 2 | "Winter" | false | "Roastbeef" | - |
| 3 | "Spring" | false | "Steak" | - |
| 4 | "Summer" | false | "Light Salad and a nice Steak" | Hey, why not!? |
| 5 | - | true | "Pasta" | - |
| + | - | - | - | - |

In this case we may want to consider guests that are vegetarian. Regardless of the season, we cannot serve them any meat. Fortunately, we always have some pasta available. By combining the two input columns "Season" and "Vegetarian Guests", we have made sure that the first four rules can only evaluate to true, if the guests are not vegetarian. Rule number 5 has a "-" in the input entry that checks the season, and this means that it can be any season, as long as the guests are vegetarians, they will get pasta.

As you can see, the combination of input entries in a rule (i.e. a table row) always follow an **AND** logic: "If it's fall **and** my guests are not vegetarian, I will serve spareribs."

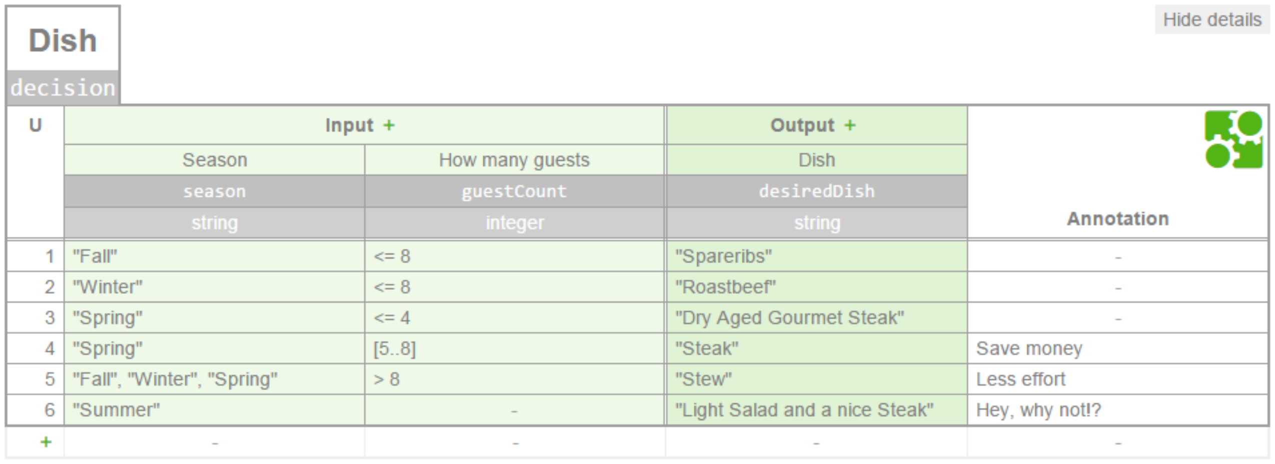
## Introducing FEEL

Now that you have a basic understanding of a decision table structure, let's take a closer look at possible input entries. It's quite simple to say that certain data should be compared to certain strings (like the fact, that the season should be summer). But DMN offers more advanced concepts for checking input entries. Part of the DMN standard is the **Friendly Enough Expression Language (FEEL)**.

FEEL defines a syntax for expressing conditions that input data should be evaluated against. For example, you can describe in FEEL that a certain input data should be

- a concrete string (like the season, that should be "summer")
- true or false (like the fact that our guests are vegetarians)
- a number that is below, above or the exact same like another given number
- a number that is between a minimum and a maximum given number
- a date that is before, later or the same like another given date
- ...and much more

To get a first idea, please have a look at the example below:



**Dish**
decision

| U | Input + | | Output + | |
|---|---|---|---|---|
| | Season | How many guests | Dish | |
| | season | guestCount | desiredDish | |
| | string | integer | string | Annotation |
| 1 | "Fall" | <= 8 | "Spareribs" | - |
| 2 | "Winter" | <= 8 | "Roastbeef" | - |
| 3 | "Spring" | <= 4 | "Dry Aged Gourmet Steak" | - |
| 4 | "Spring" | [5..8] | "Steak" | Save money |
| 5 | "Fall", "Winter", "Spring" | > 8 | "Stew" | Less effort |
| 6 | "Summer" | - | "Light Salad and a nice Steak" | Hey, why not!? |
| + | - | - | - | - |

The first thing you'll notice are two additional rows with grey cells. These rows describe technical details that the decision engine needs in order to execute the decision. The first one contains expressions that - in this case - simply refer to variable names, namely season, guestCount and desiredDish. The second one tells the engine the type of the respective outcome of the expression, in this case string and integer.

In the first examples those rows were hidden, in order to not overwhelm you right upfront. But in fact, those types are important, because they determine which FEEL expressions are available for the input entries.

Let's look at each rule, i.e. at each row:

1. If it's fall and you expect up to 8 guests, you will prepare spareribs.
2. If it's winter and you expect up to 8 guests, you will serve them roastbeef.
3. If it's spring and you expect up to 4 guests, you will indulge them with very fine, dry aged beefsteak.
4. If it's spring and you expect 5 to 8 guests, you will serve them an ordinary steak.
5. If it's fall, winter or spring, and you expect more than 8 guests, you will go for stew.
6. If it's summer, there will be a light salad and, of course, a nice steak, no matter what. Yay!

As you probably guess already, this is just the tip of the iceberg. There is much more that you can express in DMN decision tables, as we describe in the DMN Reference Guide (https://docs.camunda.org/manual/reference/dmn11/?__hstc=252030934.67bd7670c5998ed387198b9bfbdbb221.1553011190260.1553011190260.1553011190260.1&__hssc=252030934.3.1553011

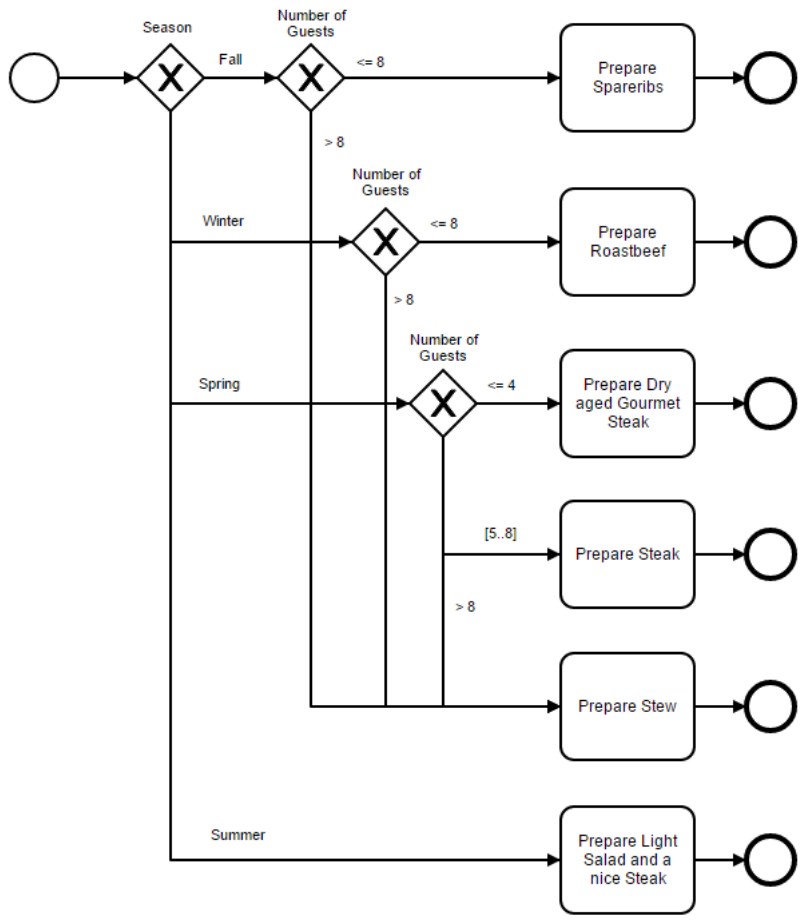
**For Developers:** You can find an implemented version of this example (https://github.com/camunda/camunda-bpm-examples/tree/master/dmn-engine/dmn-engine-java-main-method) on GitHub.

## DMN and BPMN Processes

Perhaps you're thinking:

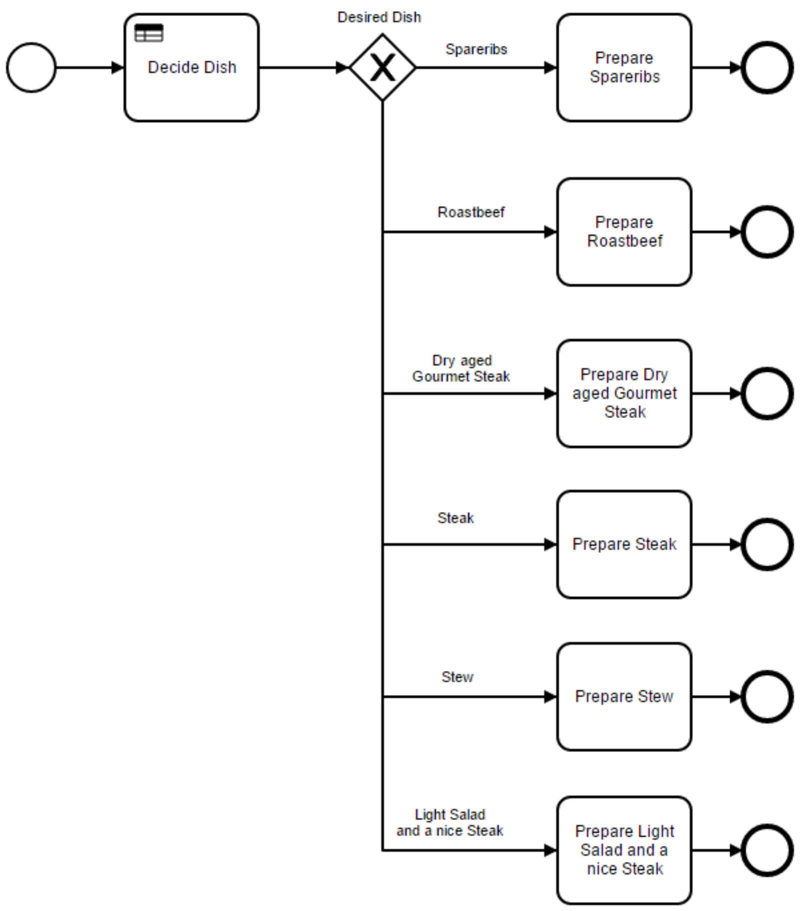
> Hey, why should I use DMN anyway, I can express those rules with BPMN gateways (https://camunda.com/bpmn/reference/#gateways-data-based-exclusive-gateways)!

If we express the example above in BPMN, it looks like this:



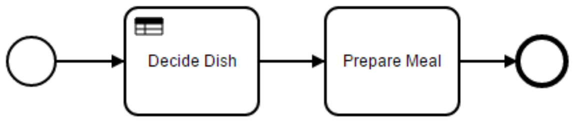
The sorrow is obvious: It's way more verbose to express rules in BPMN, especially when there are several conditions to consider. The diagram becomes complex and hard to maintain.

That is why BPMN includes a so-called **business rule task**, which should better be named **decision task** in a later version of the BPMN standard: That task refers to a decision that needs to be made, and the outcome of the decision allows the subsequent exclusive gateway to route the flow, as you can see in the example below.



During modeling as well as execution, we can link the task "Decide Dish" to the DMN decision table, that will be executed when the decision should be made, and the result will then determine the further flow in BPMN.

In this particular example, you could question the use of the flow routing anyway. There are six tasks that are about preparing a meal, the only difference being the kind of meal. There is not apparant advantage of having those six distinct tasks. An alternative pattern would be below:



It's too easy, right? But in this case, it's in fact an appropriate pattern.

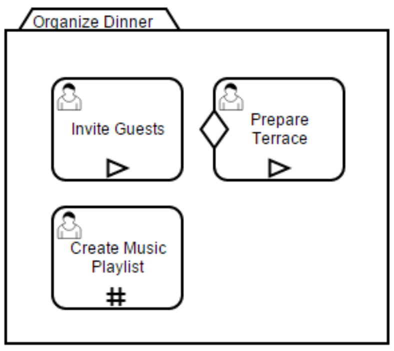
Combining BPMN with DMN is a very reasonable approach. Unfortunately, it is not yet standardized by OMG. This means, that a reference from a BPMN business rule task to a DMN decision is always vendor specific.

How Camunda links BPMN to DMN (https://docs.camunda.org/manual/develop/user-guide/process-engine/decisions/?__hstc=252030934.6

.

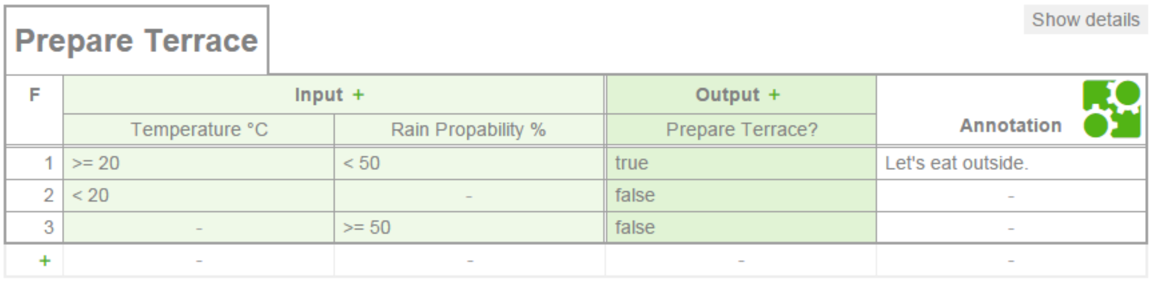
## DMN and CMMN Cases

BPMN is great for processes that are structured, but not for less structures activities. This is where CMMN kicks in. Again, it makes a lot of sense to combine this OMG standard with DMN. For example like this:



Preparing a nice dinner with friends is an art of its own, which demands a real knowledge worker.

In this CMMN case, we will need to invite our guests, for obvious reasons. We **might** need to prepare the terrace for eating outside. This is determined by the entry criterion (the little diamond on the left edge of the human task), that points to a sentry, where a decision table result is evaluated. The decision table could look like this:


Show details

Prepare Terrace

| F | Input + | | Output + | Annotation |
|---|---|---|---|---|
| | Temperature °C | Rain Propability % | Prepare Terrace? | |
| 1 | >= 20 | < 50 | true | Let's eat outside. |
| 2 | < 20 | - | false | - |
| 3 | - | >= 50 | false | - |
| + | - | - | - | - |

You may notice that the hit policy in this example is not "unique", but "first" (marked as "F"). This means that the decision engine will evaluate the rules and stop evaluating once it has found a rule that applies. In this case this makes sense, because the rules 2 and 3 both apply, if it is colder than 20° C **and** the rain propability is 50% or higher. Setting the hit policy to "unique" would therefore not be correct.

As with BPMN, the OMG has not yet standardized the way that CMMN and DMN can be combined. The example in this tutorial is therefore based on a proprietary extension that Camunda offers.

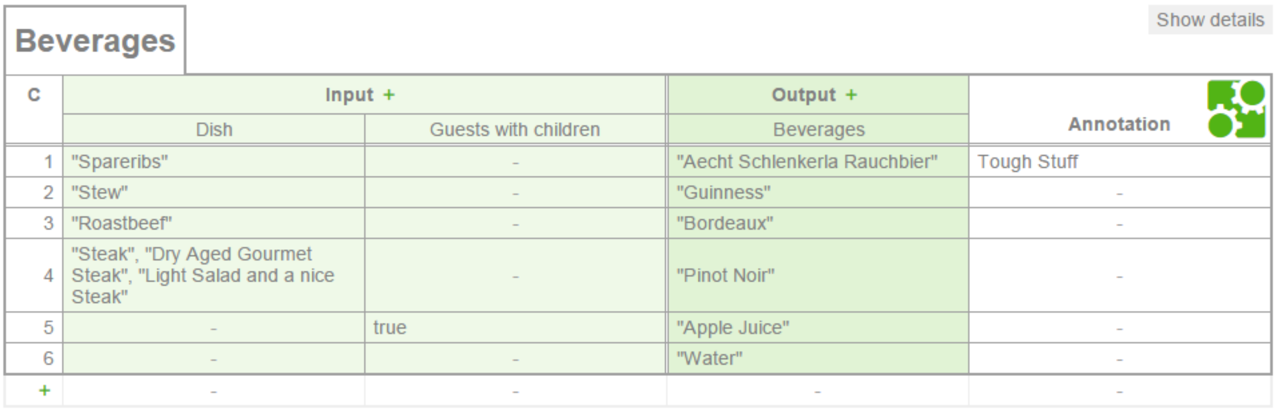
## Decision Requirements Diagrams

If you want to discuss and analyze complex decisions, that may be composed of other decisions, decision requirements diagrams (DRD) can be helpful. This is a quite simple notation defined in the DMN standard, that basically concists of

- **Decisions**: The act of determining an output value from a number of input values, using logic definition. This decision logic is what you can express in decision tables.
- **Input Data**: The input data that you "feed" into your decision logic in order to determine the output value.
- **Relations between Decisions**: You can connect decisions with arrows and therefore indicate which decision output will be considered as an input for another decision.

There are a few more symbols in the DRD notation, however the most relevant ones are these three. We should look at an example:

Let's assume that for our dinner, we also need to decide the beverages we want to serve. This decision should be based on the dish we will prepare and also consider children. The decision table could look like this:


Show details

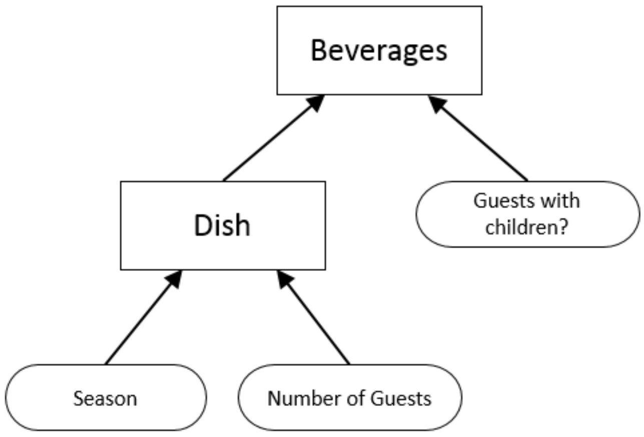
Beverages

| C | Input + | | Output + | Annotation |
|---|---|---|---|---|
| | Dish | Guests with children | Beverages | |
| 1 | "Spareribs" | - | "Aecht Schlenkerla Rauchbier" | Tough Stuff |
| 2 | "Stew" | - | "Guinness" | - |
| 3 | "Roastbeef" | - | "Bordeaux" | - |
| 4 | "Steak", "Dry Aged Gourmet Steak", "Light Salad and a nice Steak" | - | "Pinot Noir" | - |
| 5 | - | true | "Apple Juice" | - |
| 6 | - | - | "Water" | - |
| + | - | - | - | - |

You will notice that this table has a "C" in the upper left corner, instead of the "U" you have seen in the previous examples. The C stands for **Collect**, which is an other **hit policy**, and it means that more than one rule could be true, which would lead to a list of output values.

For example, if we will have spareribs, and our guests come with children, we will serve water, apple juice and the famous Aecht Schlenkerla Rauchbier (http://www.schlenkerla.de/indexe.html).

Obviously, we need to determine the dish we will prepare, before we can decide the beverages. And this relation is what you can describe in a DRD, like we did in this example:


Beverages
Dish
Guests with children?
Season
Number of Guests